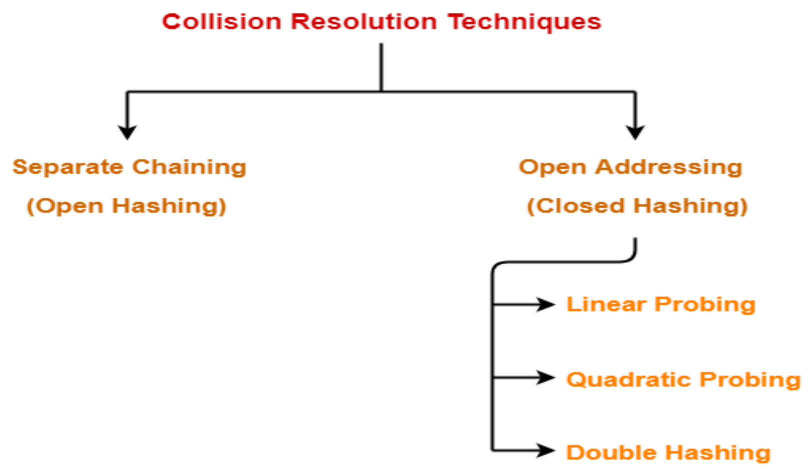
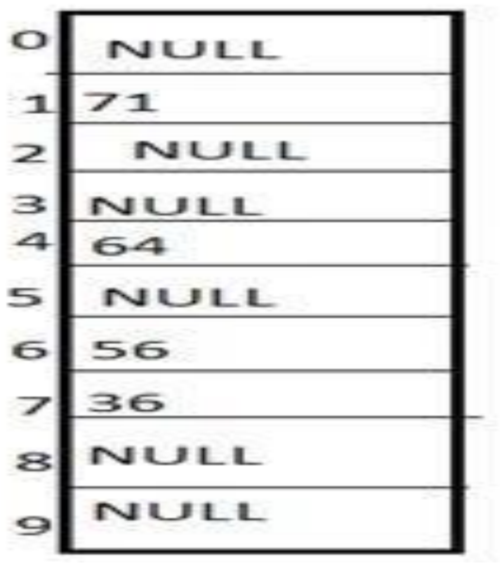


LINEAR PROBING

It is very easy and simple method to resolve or to handle the collision. In this collision can be solved by placing the second record linearly down, whenever the empty place is found. In this method there is a problem of clustering which means at some place block of a data is formed in a hash table.

Example: Let us consider a hash table of size 10 and hash function is defined as H(key)=key % table size. Consider that following keys are to be inserted that are 56,64,36,71.

In this diagram we can see that 56 and 36 need to be placed at same bucket but by linear probing technique the records linearly placed downward if place is empty i.e. it can be seen 36 is placed at index 7.

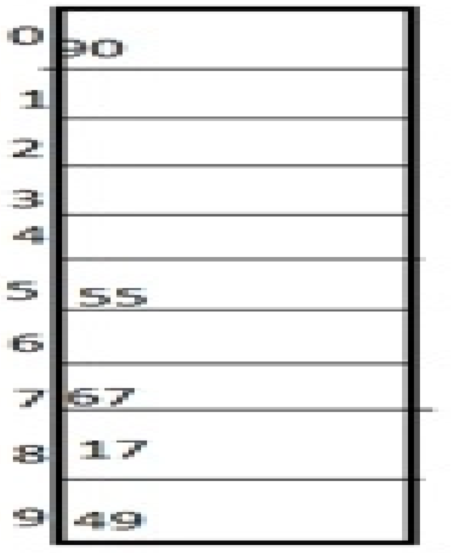


0	NULL
1	71
2	NULL
3	NULL
4	64
5	NULL
6	56
7	36
8	NULL
9	NULL

QUADRATIC PROBING

This is a method in which solving of clustering problem is done. In this method the hash function is defined by the H(key)=(H(key)+x*x)%table size. Let us consider we have to insert following elements that are:-67, 90,55,17,49.

In this we can see if we insert 67, 90, and 55 it can be inserted easily but at case of 17 hash function is used in such a manner that :-(17+0*0)%10=7 (when x=0 it provide the index value 7 only) by making the increment in value of x. let x =1 so (7+1*1)%10=8.in this case bucket 8 is empty hence we will place 17 at index 8.



DOUBLE HASHING

It is a technique in which two hash function are used when there is an occurrence of collision. In this method 1 hash function is simple as same as division method. But for the second hash function there are two important rules which are
1.It must never evaluate to zero.

2.Must sure about the buckets, that they are probed.

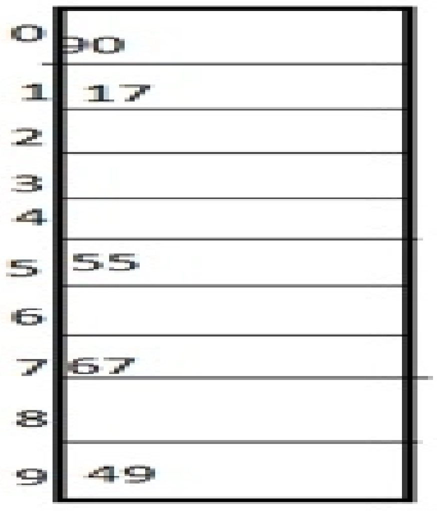
The hash functions for this technique are:
H1(key)=key % table size
H2(key)=P-(key mod P)

Where, **p**is a prime number which should be taken smaller than the size of a hash table.

Example: Let us consider we have to insert 67, 90,55,17,49.

In this we can see 67, 90 and 55 can be inserted in a hash table by using first hash function but in case of 17 again the bucket is full and in this case we have to use the second hash function which is H2(key)=P-(key mode P) here p is a prime number which should be taken smaller than the hash table so value of p will be the 7.

i.e. H2(17) = 7-(17%7) = 7-3 = 4 that means we have to take 4 jumps for placing the 17. Therefore 17 will be placed at index 1.



Example of Double Hashing in Data Structure

The idea behind double hashing is fairly simple,

1. Take the key you want to store on the hash-table.
2. Apply the first hash function *h*1*h*1(key) over your key to get the location to store the key.
3. If the location is empty, place the key on that location.
4. If the location is already filled, apply the second hash function *h2(key)* in combination with the first hash function *h1(key)* to get the new location for the key.

Let’s explore this idea in more detail, with a hands-on problem,

Problem Statement:

Insert the keys 79, 69, 98, 72, 14, 50 into the **Hash Table of size 13**. Resolve all collisions using Double Hashing where first hash-function is ***h*1(k) = k mod 13** and second hash-function is ***h*2*h*2(k) = 1 + (k mod 11)**

Solution:

Initially, all the hash table locations are empty. We pick our first key = 79 and apply *h*1*h*1(k) over it,

***h*1(79) = 79 % 13 = 1, hence key 79 hashes to 1*st* location of the hash table. But before placing the key, we need to make sure the 1*st* location of the hash table is empty. In our case it is empty and we can easily place key 79 in there.**

Second key = 69, we again apply *h*1(k) over it, *h*1(69) = 69 % 13 = 4, since the 4*th* location is empty we can place 69 there.

Third key = 98, we apply *h*1*h*1(k) over it, *h*1*h*1(98) = 98 % 13 = 7, 7*th* location is empty so 98 placed there.

Fourth key = 72, *h*1(72) = 72 % 13 = 7, now this is a collision because the 7*th* location is already occupied, we need to resolve this collision using double hashing.

***hnew* = [*h*1*h*1(72) + i * *h*2*h*2(72)] % 13**

= [7 + 1 * (1 + 72 % 11)] % 13

= 1

Location 1*st* is already occupied in the hash-table, hence we again had a collision. Now we again recalculate with i = 2

***hnew* = [*h*1*h*1(72) + i * *h*2*h*2(72)] % 13**

= [7 + 2 * (1 + 72 % 11)] % 13

= 8

Location 8*th* is empty in hash-table and now we can place key 72 in there.

Fifth key = 14, $h1h1(14)$ = 14%13 = 1, now this is again a collision because 1st location is already occupied, we now need to resolve this collision using double hashing.

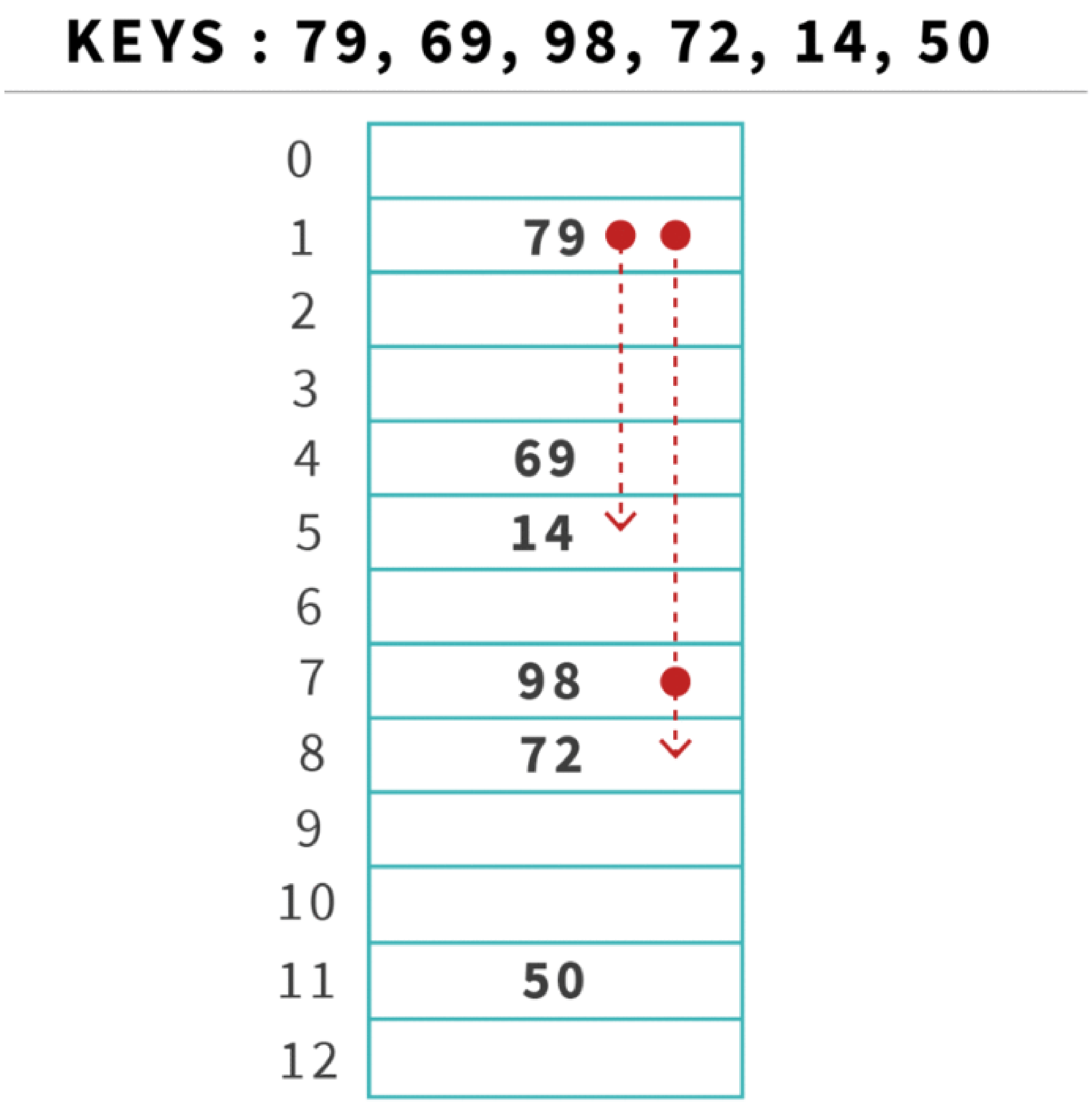
$hnew$ = [$h1h1(14)$ + i * $h2h2(14)$] % 13

= [1 + 1 * (1 + 14 % 11)] % 13

= 5

Location 5*th* is empty and we can easily place key 14 there.

Sixth key = 50, $h1h1(50)$ = 50%13 = 11, 11*th* location is already empty and we can place our key 50 there.



Why Use Double Hashing?

Double Hashing is one of the popular collision resolution techniques. The other popular variants which serve the same purpose are Linear **Probing** and **Quadratic Probing**. But if other techniques are available, then why do we need double hashing in the first place?

Double Hashing offers better resistance against clustering. A major reason for this is the use of dual functions. Dual functions enable double hashing to perform a more uniform distribution of keys, resulting in lower collisions.

Advantages of Double Hashing in Data Structure

1. After each collision, we recomputed the new location for the element in the hash-table. For successive collisions, this generates a sequence of locations. If the sequences are unique then the number of collisions are less.
2. Double Hashing creates most unique sequences, providing a more uniform distribution of keys within the hash-table.
3. If the hash function is not good enough, the elements tend to form grouping in the hash-table. This problem is known as clustering. Double Hashing is least prone to clustering.

Practice Problem Based on Double Hashing

Problem Statement 1:

Given the two hash functions, **$h1h1$(k) = k mod 23** and **$h2h2$(k) = 1 + k mod 19**. Assume the table size is 23. Find the address returned by double hashing after 2nd collision for the key = 90.

Solution:

We will use the formula for double hashing-

h(k,i) = ($h1h1$(k) + i * $h2h2$(k))%m

As it is given, k = 90, m = 23

Since the 2nd collision has already occurred, **i = 2**. Substituting the values in the above formula we get,

h(90,2) = [(90 % 23) + 2 * (1 + 90 % 19)] % 23

= [21 + 2 * 15] % 23

= 5

Hence after the second collision, the address returned by double hashing for Key = 90 is 5.